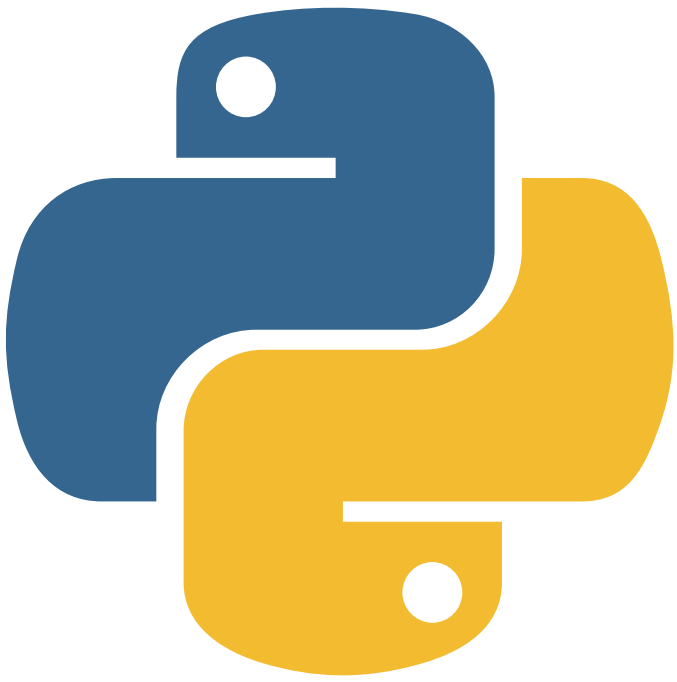
Day 5 Python Basics to Advanced

Day 5 : Control Flow

Today's Topics:

- **If statements?**
- **Else and Elif statements**
- **Nested If statement**
- **The While Loop**
- **The For Loop**
- **Loop control statements**
- **Use range() in For Loop**



By @Curious_.programmer

5. Control Flow

Made By:

Yadneyesh (Curious Coder)
CodWithCurious.com

Control Flow

Control flow directs program execution through structures like loops, conditionals, and functions, determining the order and path of operations.

If statements

An `if` statement in Python checks whether a condition is true or false. If the condition is true, the code inside the `if` block runs. If false, the code is skipped. It's used to make decisions in the program, executing specific actions based on conditions

example:

```python
age = 18

if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

Else and elif statements

In Python, else and elif statements are used alongside if to handle multiple conditions and alternative actions.

- elif (else if): Checks another condition if the previous if was false. You can have multiple elif statements.
- else: Runs when none of the if or elif conditions are true. It's the "default" action.

example:


```
temperature = 75

if temperature > 85:
    print("It's too hot outside!")
elif temperature > 70:
    print("It's a nice day.")
elif temperature > 50:
    print("It's a bit chilly.")
else:
    print("It's cold outside.")
```

Nested if Statement

A nested `if` statement in Python is an `if` statement inside another `if`. It lets you check multiple related conditions in sequence.

For example:

If you first check the weather and it's sunny, you can then check how many guests are coming. Depending on the number of guests, you decide between different activities, like a barbecue or picnic.

If the weather isn't sunny, you skip the nested checks and go straight to an alternative action, like staying indoors.

example:


```
temperature = 78
humidity = 65

if temperature > 70:
    print("The weather is warm.")

    if humidity > 60:
        print("It's also quite humid.")
    else:
        print("The humidity is low.")
else:
    print("The weather is cool.")

    if humidity > 60:
        print("It's humid despite the cool temperature.")
    else:
        print("The weather is cool and dry.")
```

The While Loop

A `while` loop in Python repeatedly executes a block of code as long as a specified condition is true.

It first checks the condition; if true, the code inside runs. After each iteration, the condition is rechecked. The loop continues until the condition becomes false.

For example

A `while` loop can keep counting up as long as the count is below a certain number. It's useful for scenarios where you don't know in advance how many iterations are needed.


```python
# Initialize the counter
count = 0

# Start the while loop
while count < 5:
    print("Count is:", count)
    count += 1  # Increment the counter

print("Loop ended.")
```

The For Loop

A `for` loop in Python is used to iterate over a sequence, such as a list, tuple, or range, executing a block of code for each item in the sequence.

Unlike a `while` loop, which runs until a condition is false, a `for` loop runs a set number of times based on the length of the sequence.

For example

It can go through a list of numbers, processing each one in turn. It's ideal for repetitive tasks like iterating over data collections.

example:


```python
# List of items
fruits = ["apple", "banana", "cherry"]

# For loop to iterate over the list
for fruit in fruits:
    print(fruit)

print("Loop ended.")
```

Loop Control Statements

Loop control statements in Python allow you to alter the flow of a loop’s execution. They include:

- `break`: This statement immediately exits the loop, regardless of whether the loop's condition is still true. It’s useful for stopping a loop when a specific condition is met, like when searching for an item in a list and finding it before the loop has iterated through the entire list.

example:

```
for i in range(10):
    if i == 5:
        break  # Exit the loop when i is 5
    print("Current number:", i)

print("Loop ended.")
```


- 'continue': This statement skips the rest of the current loop iteration and proceeds to the next iteration. It’s helpful for bypassing certain parts of the loop based on a condition, like skipping even numbers in a loop that processes a range of numbers.

example:

```python
for i in range(10):
    if i % 2 == 0:
        continue  # Skip even numbers
    print("Odd number:", i)

print("Loop ended.")
```

Using Range() in For Loop

The range() function in Python is commonly used in for loops to iterate over a sequence of numbers.


```python
# For loop using range to print numbers from 0 to 4
for i in range(5):
    print("Number:", i)
```

Here's a basic rundown of how it works:

- start: The starting value of the sequence (inclusive). If omitted, it defaults to 0.
- stop: The ending value of the sequence (exclusive). The loop will run until it reaches this value.
- step: The amount by which the sequence is incremented. If omitted, it defaults to 1.

You can use range() in a for loop for various tasks like iterating through lists, generating sequences of numbers, or performing repetitive actions a specific number of times.

example:

```
# For loop using range to print numbers from 10 to 1 in reverse
for i in range(10, 0, -1):
    print("Number:", i)
```


6 : Lists and Tuples

Will Post Follow For More

Want This PDF?

Join Our Telegram and Download 100+ PDFs and Handwritten Notes